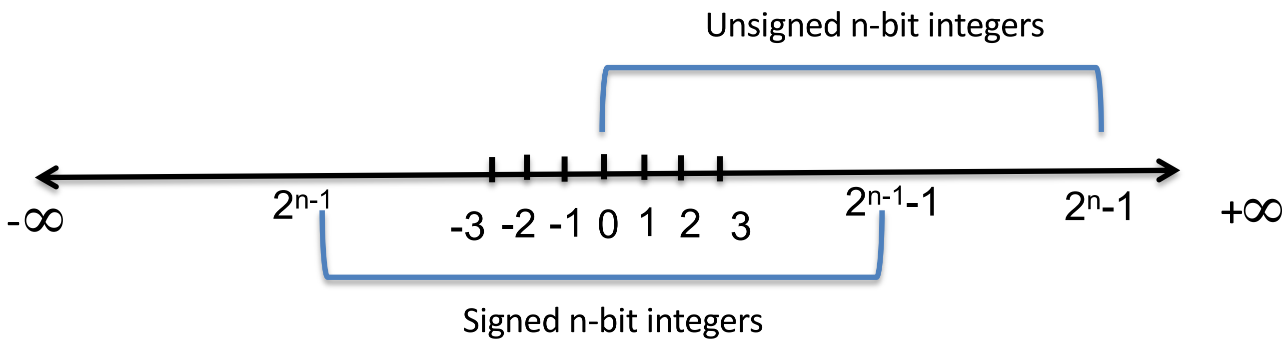
Floating point

Jinyang Li

Floating Point (FP) lesson plan

- Normalized binary exponential notation
- Strawman 32-bit FP
- IEEE FP format
- Rounding

Previously…



What about real numbers?

Represent real numbers: the decimal way

Real Number	Decimal Representation
11 / 2	$(5.5)_{10}$
1 / 3	$(0.3333333...)_{10}$
√2	$(1.4128...)_{10}$

$$(1.4128...)_{10} = 1 * 10^0 + 4 * 10^{-1} + 1 * 10^{-2} + 2 * 10^{-3} + ...$$

Binary Representation

$(5.5)_{10} = 4 + 1 + 1/2 = 2^2 + 2^0 + 2^{-1}$

$= (101.1)_2$

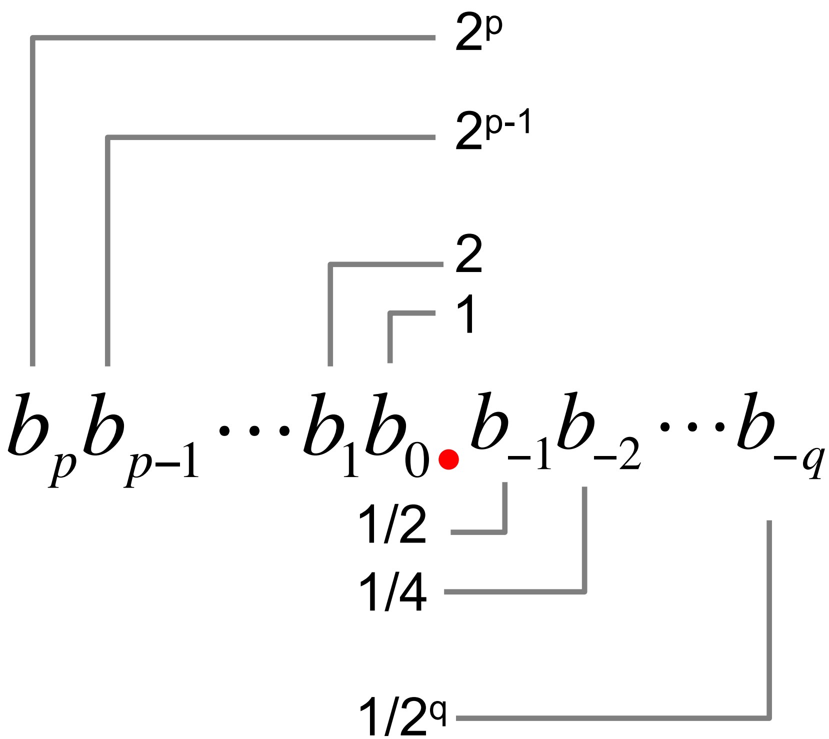
Binary Representation

$$(0.1)_{10} = 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + \dots$$

$$= (0.0001100110011...)_2$$

Binary Representation

$$b_p b_{p-1} \cdots b_1 b_0 . b_{-1} b_{-2} \cdots b_{-q} = \sum_{i=-q}^{p} 2^i \times b_i$$



Binary representation

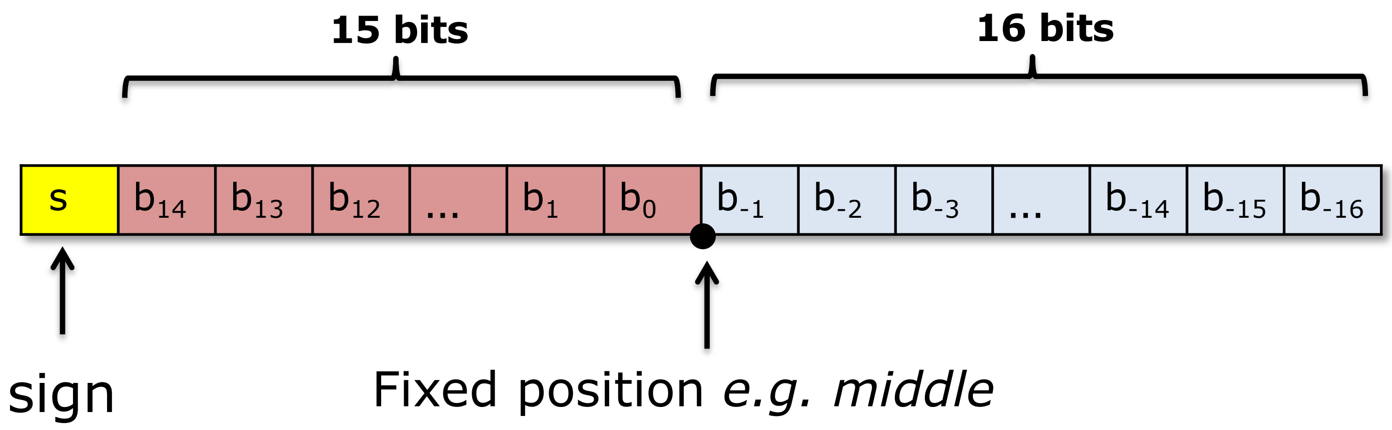
What's the decimal value of $(10.01)_2$

Binary representation

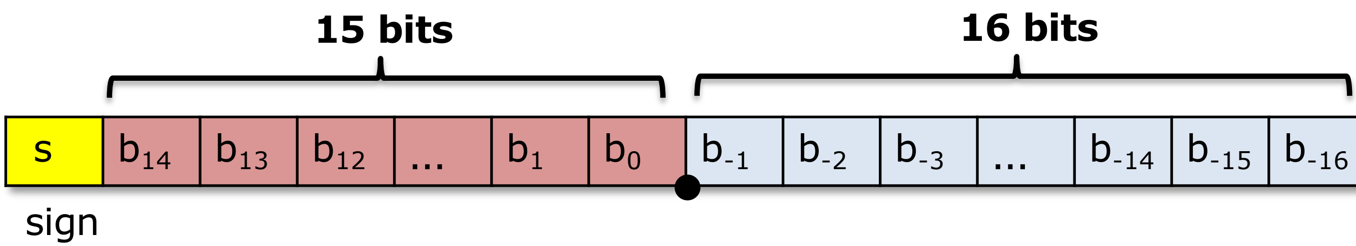
What's the decimal value of $(10.01)_2$

Answer: 2.25

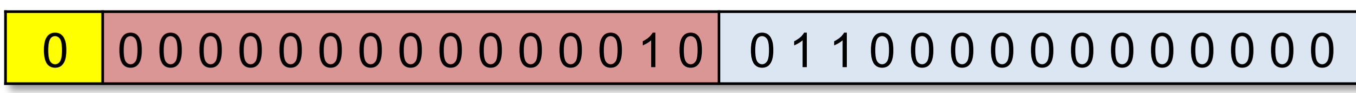
Making the representation fixed width
Strawman: fixed point



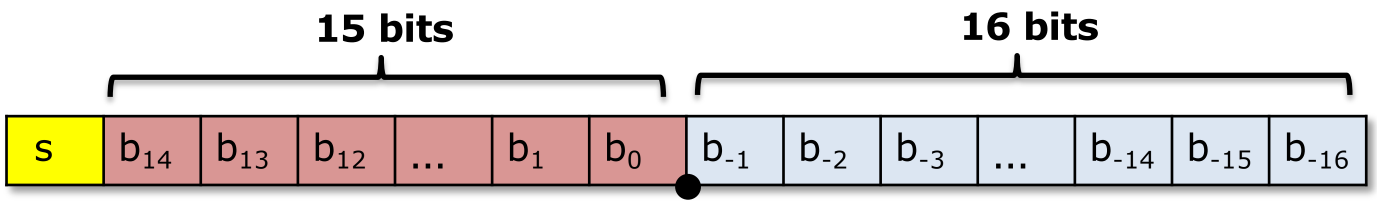
Fixed point representation



Example: $(10.011)_2$

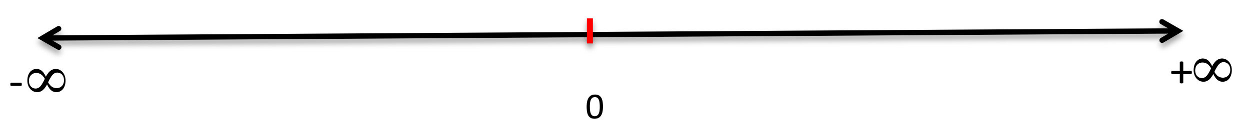


Problems of Fixed Point

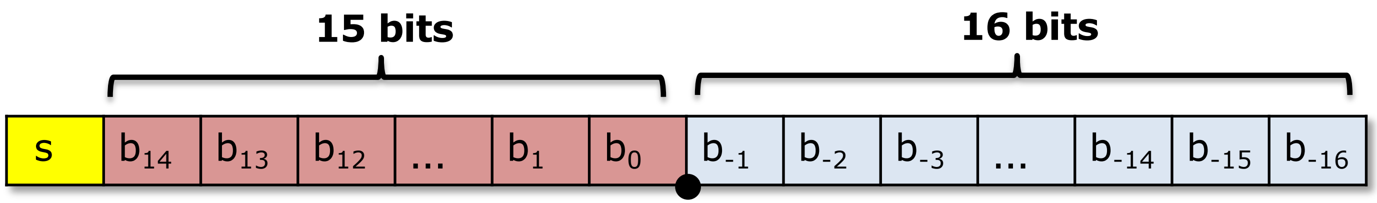


Range?

Precision?



Problems of Fixed Point



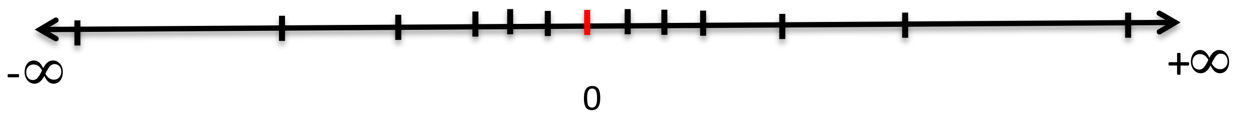
– Limited range and precision: e.g., 32 bits
  - Range: $[-2^{15}+2^{-16}, 2^{15}-2^{-16}]$
  - Highest precision: 2^{-16}

→Rarely used (No built-in hardware support)

Floating point: key idea

- Limitation of fixed point:
  - Even spacing results in hard tradeoff between high precision and high magnitude

- How about un-even spacing between numbers?



Floating Point: decimal

Based on exponential notation (aka normalized scientific notation)

$r_{10} = \pm M * 10^{E}$, where $1 <= M < 10$

M: significant (mantissa), E: exponent

Floating Point: decimal

Example:

```
365.25 = 3.6525 * 10^2
0.0123 = 1.23 * 10^-2
```

Decimal point **floats** to the position immediately after the first nonzero digit.

Floating Point: binary

Binary exponential representation

$\pm M * 2^E$, where $1 <= M < 2$

$M = (1.b_1b_2b_3\ldots b_n)_2$

M: significant, E: exponent

$(5.5)_{10} = (101.1)_2 = (1.011)_2 * 2^2$

Floating Point

Binary exponential representation

$\pm M * 2^E$, where $1 <= M < 2$

$M = (1.b_1b_2b_3 \ldots b_n)_2$

} Also called normalized representation

M: significant, E: exponent

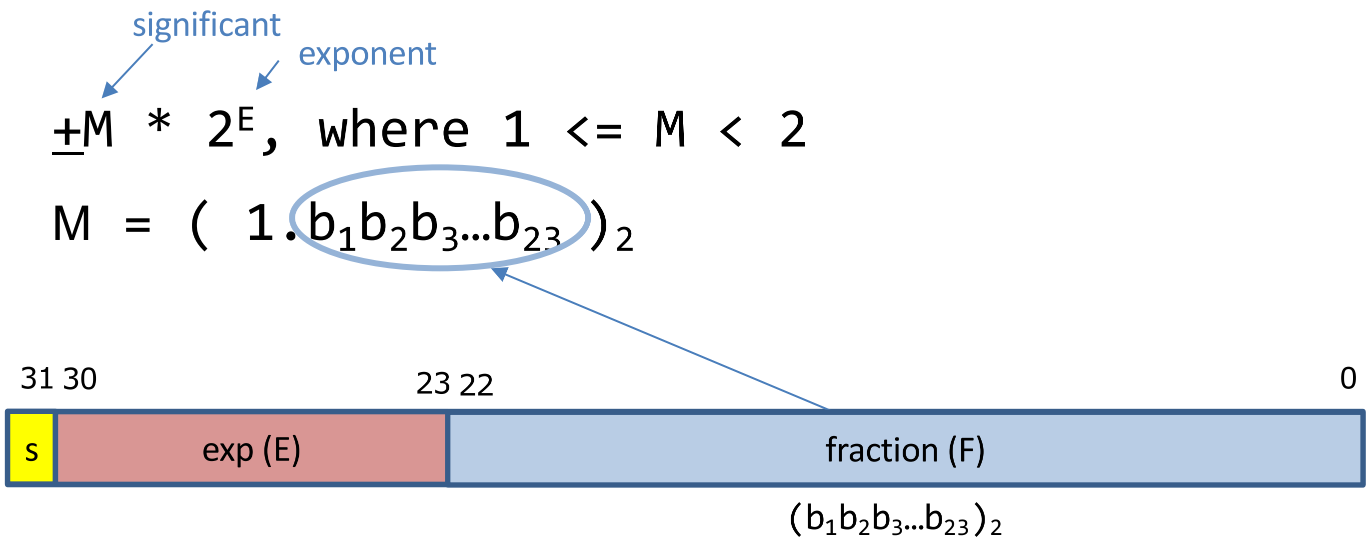
$(5.5)_{10} = (101.1)_2 = (1.011)_2 * 2^2$

(Binary) normalized representation of $(10.25)_{10}$?

(Binary) normalized representation of $(10.25)_{10}$?

Answer: $(10.25)_{10} = (1010.01)_2 = (1.01001)_2 * 2^3$

Strawman FP: normalized representation in 32-bit



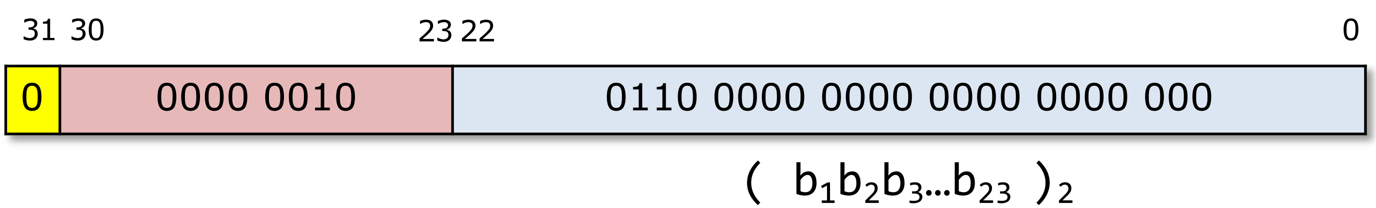
Strawman 32-bit FP: Example

significant exponent

$\pm M * 2^E$, where $1 <= M < 2$

$M = (1.b_1b_2b_3 \ldots b_{23})_2$

Example: $(5.5)_{10} = (101.1)_2 = (1.011)_2 * 2^2$



More Strawman 32-bit FP Examples

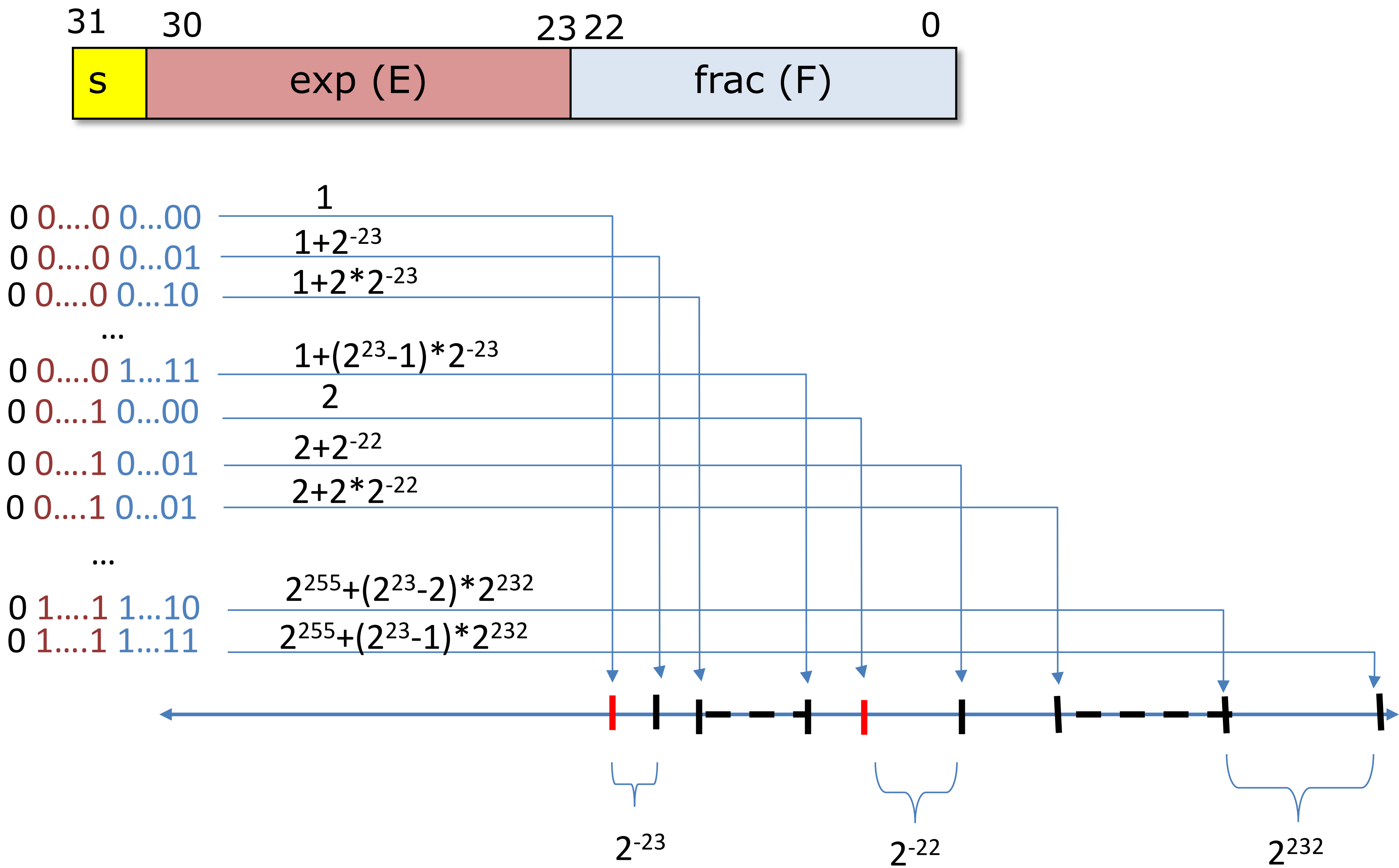
Example: $(65)_{10} = (1000001)_2 = (1.000001)_2 * 2^6$

31	30 … 23	22 … 0
0	0000 0110	0000 0100 0000 0000 0000 000

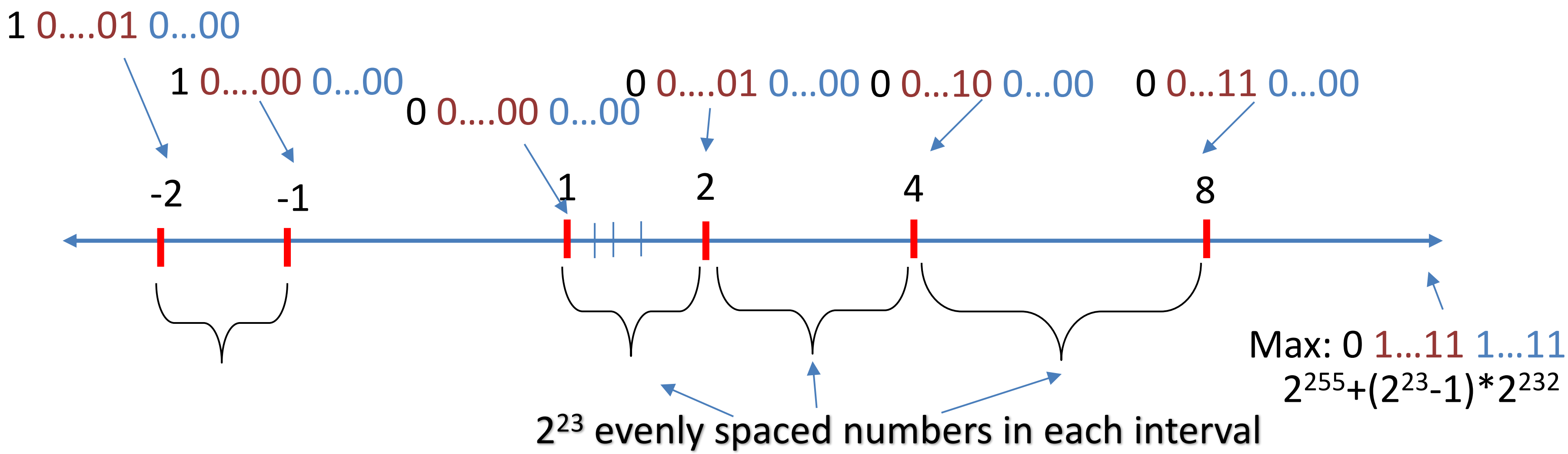
Another example: $(10.25)_{10} = (1010.01)_2 = (1.01001)_2 * 2^3$

31	30 … 23	22 … 0
0	0000 0011	0100 1000 0000 0000 0000 000

Strawman FP on a number line



Strawman 32-bit FP: pros and cons



- The good 👍
  - Large range $[1, 2^{255}+(2^{23}-1)*2^{232}]$, $[-2^{255}-(2^{23}-1)*2^{232}, -1]$
  - Allows easy comparison: compare FPs by bit patterns
- The bad 👎
  - No 0!
  - No [-1, 1]
  - Max precision (2^{-23}) not high enough
  - No representation of special cases: ∞

IEEE Floating Point Standard

- Lots of FP implementations in 60s/70s
  - Code was not portable across processors
- IEEE formed a committee (IEEE.754) to standardize FP format and specification.
  - IEEE FP standard published in 1985
  - Led by William Kahan

Prof. William Kahan
University of California at Berkeley
Turing Award (1989)

IEEE Floating Point Standard

- This class only covers basic FP materials
- A deep understanding of FP is crucial for numerical/scientific computing
  - More FP is covered in undergrad/grad classes on numerical methods

Numerical Computing with IEEE Floating Point Arithmetic

Including One Theorem, One Rule of Thumb, and One Hundred and One Exercises

Michael L. Overton
Courant Institute of Mathematical Sciences
New York University
New York, New York

Goals of IEEE Standard

- Consistent representation of floating point numbers
  - Address the limitation of our FP strawman

- Correctly rounded floating point operations, using several rounding modes.

- Consistent treatment of exceptional situations such as division by zero

IEEE FP: Carve out subsets of bit-patterns from normalized representation

$\underline{+}M * 2^E \quad M = (1.b_0b_1b_2b_3\ldots b_n)_2$

$(b_0b_1b_2b_3\ldots b_n)_2$

For normalization representation,
exp can not be $(1111\ 1111)_2$ or $(0000\ 0000)_0$

exp_{max} = ? 254, $(1111\ 1110)_2$

exp_{min} = ? 1, $(0000\ 0001)_2$

IEEE FP: Represent negative exponents using bias

$\underline{+}M * 2^E, \ M = (1.b_0b_1b_2b_3\ldots b_n)_2$

To represent FPs in (-1,1), we must allow negative exponent.

- How to represent negative E?
  - ~~2's complement~~
  - use bias

Why? Using bias instead of 2's complement allows simple comparison of FPs using their bit-patterns

31	30 … 23	22 … 0
s	exp = E + 127	fraction (F)

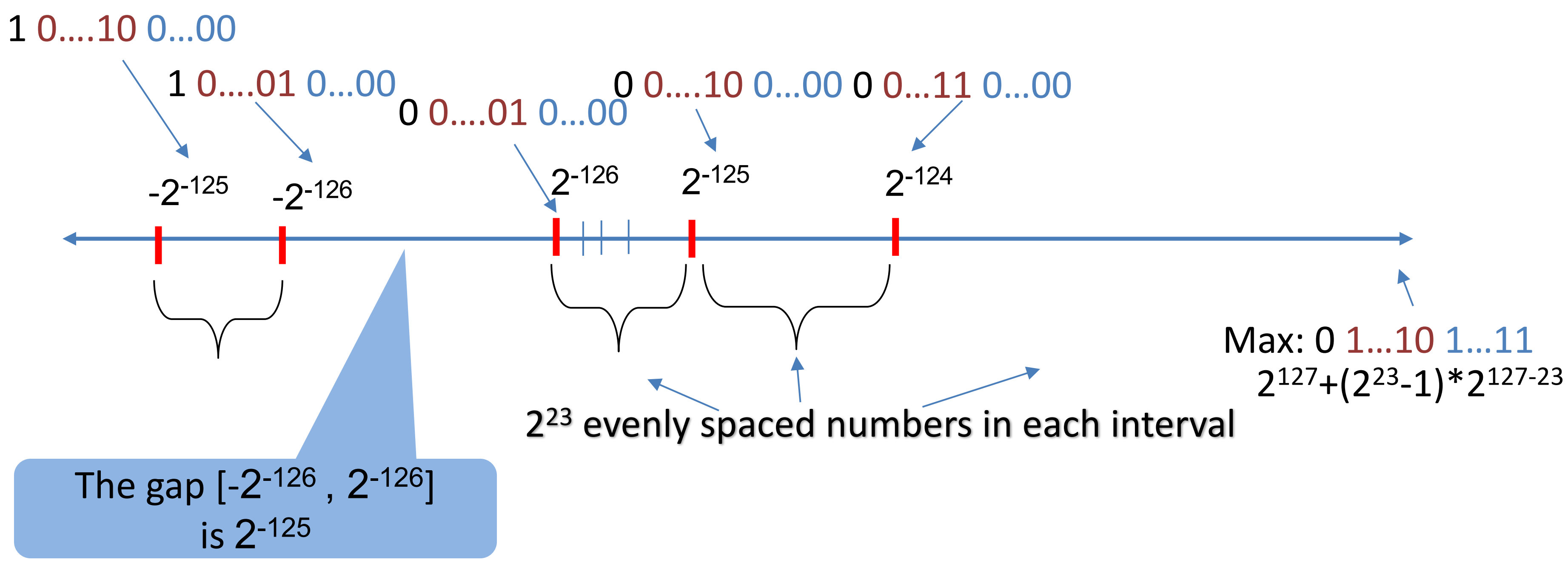
$(b_0b_1b_2b_3\ldots b_n)_2$

IEEE FP normalized representation

$\pm M * 2^E,\ M = (1.b_0b_1b_2b_3\ldots b_n)_2$

31	30 … 23	22 … 0
s	exp = E + 127	fraction (F)

fraction (F) = $(b_0b_1b_2b_3\ldots b_n)_2$



Represent values close and equal to 0

IEEE FP denormalized representation: represent values close and equal to 0

$\pm M * 2^E$

Normalized Encoding:

31	30 … 23	22 … 0
s	exp = E + 127	fraction (F)

$1 <= M < 2, M = (1.F)_2$

Denormalized Encoding:

31	30 … 23	22 … 0
s	exp =0000 0000	fraction (F)

$E = 1 – Bias = -126$

$0 <= M < 1, M = (0.F)_2$

Zeros

+0.0

0	0000 0000	0000 0000 0000 0000 0000 000

-0.0

1	0000 0000	0000 0000 0000 0000 0000 000

Denormalized FP example

Smaller than the smallest E (-126) of normalized encoding

What's the IEEE FP format of $(1.0)_2 * 2^{-127}$?

$$(1.0)_2 * 2^{-127} = (0.1)_2 * 2^{-126}$$

0	0000 0000	1000 0000 0000 0000 0000 000

What we've learnt so far

- Normalized binary representation of real numbers

Answer: $(10.25)_{10} = (1010.01)_2 = (1.01001)_2 * 2^3$

What we’ve learnt so far:
IEEE FP normalized + denormalized

31	30 … 23	22 … 0
s	exp = E + 127	fraction (F)

If (exp!=0 && exp!=255) $n = (1.F)_2 * 2^{exp-127}$ (normalized)

0	1000 0010	0100 1000 0000 0000 0000 000

$n = (1.01001)_2 * 2^{130-127}$

---

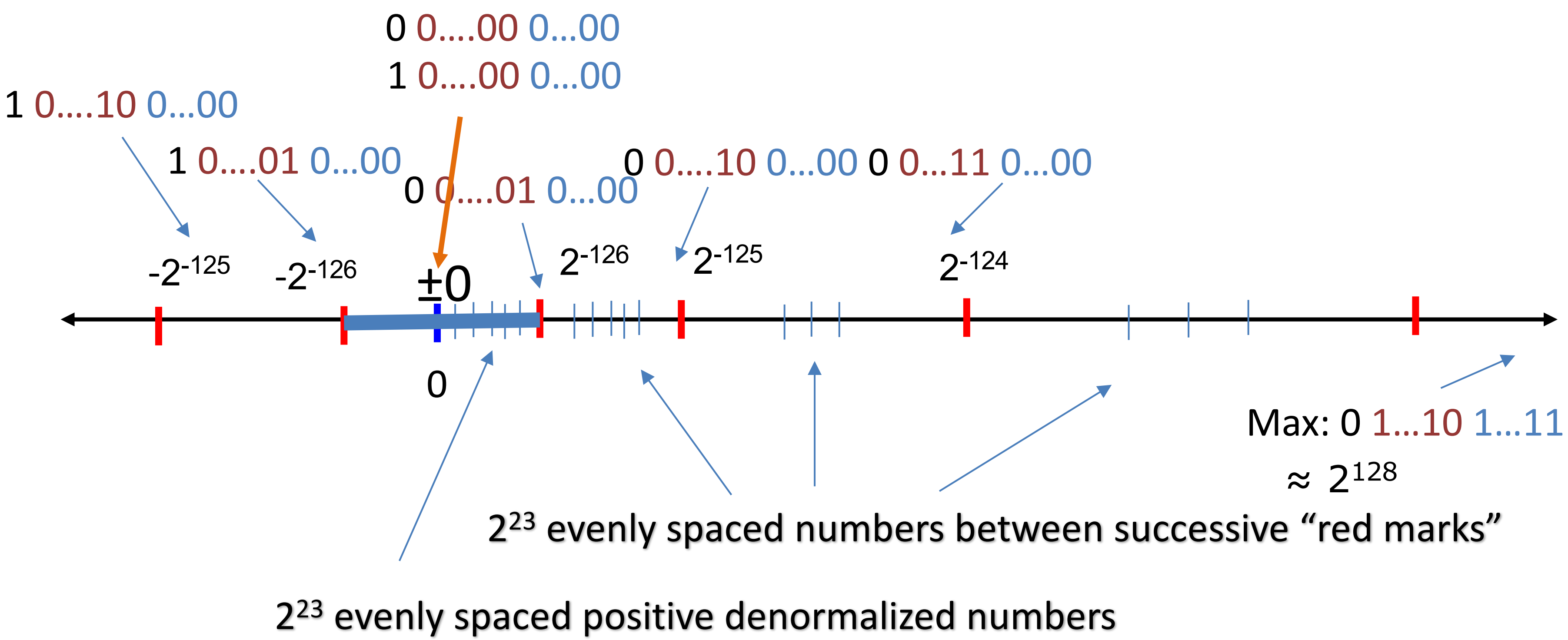
s	0000 0000	fraction (F)

If (exp == 0) $n = (0.F)_2 * 2^{-126}$ (denormalized)

0	0000 0000	0100 1000 0000 0000 0000 000

$n = (0.01001)_2 * 2^{-126}$

What we've learnt so far: IEEE FP normalized + denormalized



Precision is higher for numbers close to zero

Floating Point (cont’d) lesson plan

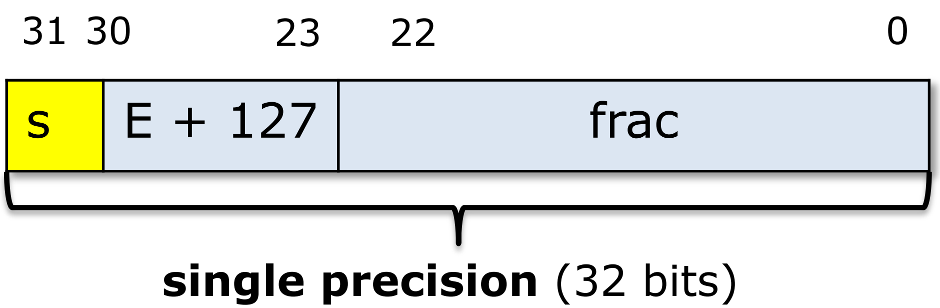
- IEEE FP special values
- Revisit FP: Toy 8-bit FP
- Rounding
- FP operations

IEEE FP: special values

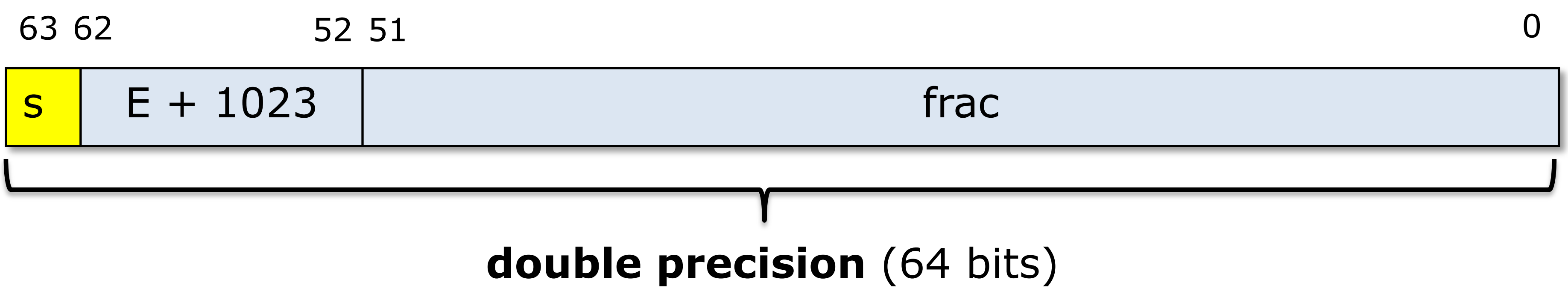
Special Value's Encoding:

values	**sign**	**frac**
+∞	0	all zeros
- ∞	1	all zeros
NaN	any	non-zero

IEEE FP: single vs. double precision



float f = 0.1;
double d = 0.1;



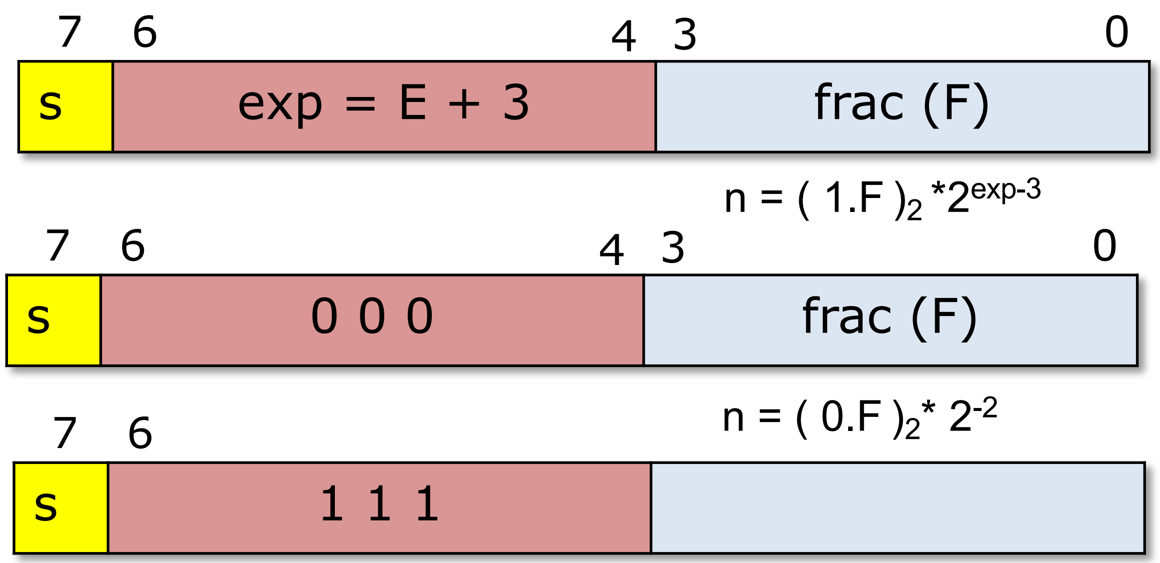
single/ double precision

	E_{min}	E_{max}	N_{min}	N_{max}
Float	-126	127	2^{-149}	$\approx 2^{128}$
Double	-1022	1023	2^{-1074}	$\approx 2^{1024}$

A toy 8-bit FP in the spirit of IEEE FP

$\underline{+}M * 2^E$

- exponent: 3 bits
- fraction: 4 bits
- **bias: 3**



Normalized encoding
exp ≠ 000, 111

$n = (1.F)_2 * 2^{exp-3}$

Denormalized encoding
exp = 000

$n = (0.F)_2 * 2^{-2}$

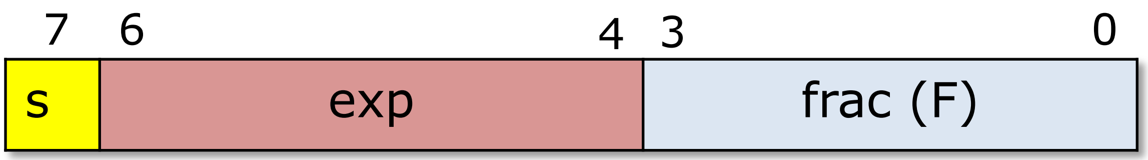
Special values encoding
exp = 111

- Smallest positive number?
- Range?
- How many distinct numbers?

A toy 8-bit FP in the spirit of IEEE FP

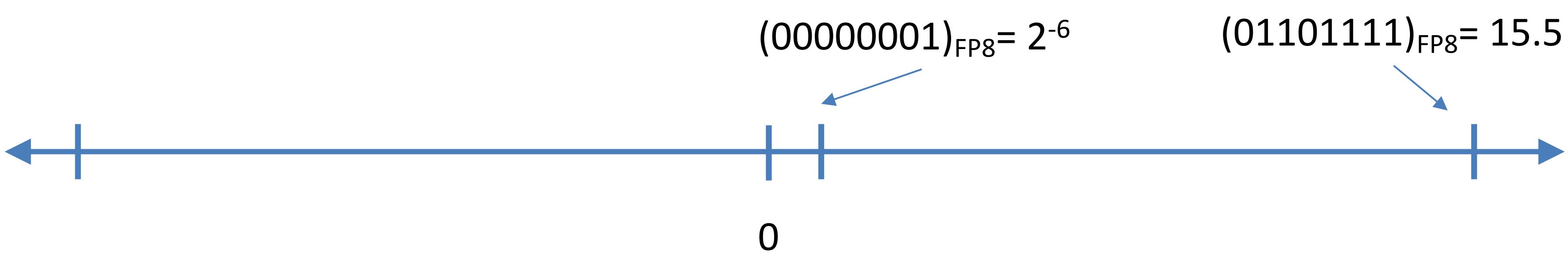
$\pm M * 2^E$

- exponent: 3 bits
- fraction: 4 bits
- **bias: 3**



If exp!=0 && exp!=$(111)_2$
$n = (1.F)_2 * 2^{exp-3}$

Else if exp == 0
$n = (0.F)_2 * 2^{-2}$

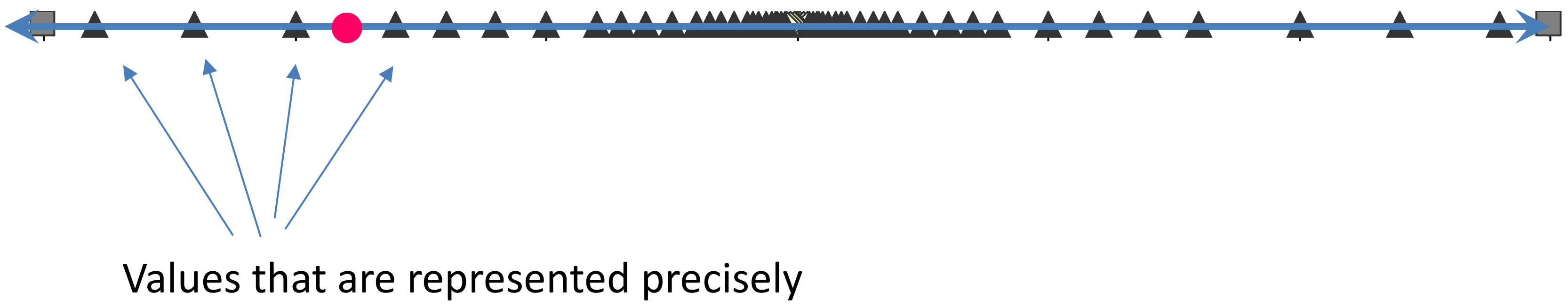


$2^8 - 2^5 - 1$ distinct numbers: there are 2^8 total bit-patterns, 2^5 special values, 0 has 2 bit-patterns.

Floating Point (cont’d) lesson plan

- IEEE FP special values
- Revisit FP: Toy 8-bit FP
- Rounding
- FP operations

FP: Rounding



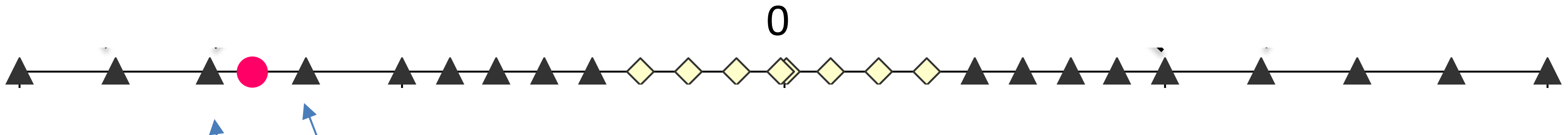
What if the result of computation is at ● ?

Rounding: Use the “closest” representable value *x’* for x.

4 modes:

- Round-down
- Round-up
- Round-toward-zero
- Round-to-nearest (Round-to-even in text book)

Round up vs. round down



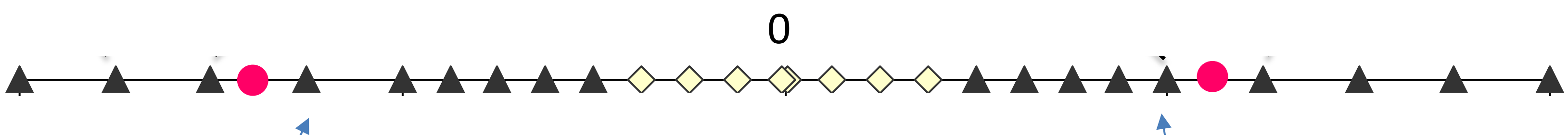
Round up rounds to the right

Round(x) = x_+ ($x_+ >= x$)

Round down rounds to the left

Round(x) = x_- ($x_- <= x$)

Round towards zero



Rounds to the right if x < 0

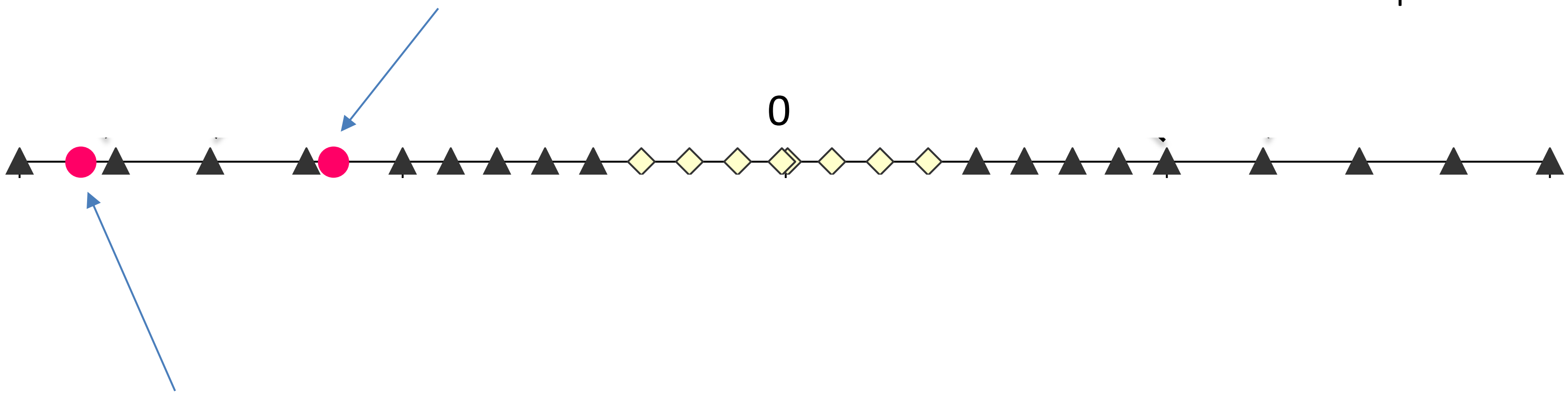
Round(x) = x_+ if x < 0

Rounds to the left if x >0

Round(x) = x_- if x < 0

Round to nearest; ties to even

Round to the left if x_- is nearer to x than x_+



Round to the right if x_+ is nearer to x than x_-

In case of a tie, the one with its least significant bit equal to zero is chosen.

How does CPU know if some 4-byte value should be interpreted as IEEE FP or integers?

CPU uses separate registers for floating point and ints.

CPU uses different instructions for floating points and int operations.

Floating Point (cont'd) lesson plan

- IEEE FP special values
- Revisit FP: Toy 8-bit FP
- Rounding
- FP operations

Floating point operations

- FP Caveats:
  - Invalid operation: 0/0, sqrt(-1), $\infty + \infty$
  - Divide by zero: $x/0 \rightarrow \infty$
  - Overflows: result too big to fit
  - Underflows: 0 < result < smallest denormalized value
  - Inexact: round it!
- FP addition: commutative but not always associative
- FP multiplication: commutative but not always associative and distributive

Floating point in real world

- Storing time in computer games as a FP?
- Precision diminishes as time gets bigger

FP value (decimal)	Time value	FP precision	Time precision
1	1 sec	1.19E-07	119 nanoseconds
100	~1.5 min	7.63E-06	7.63 microseconds
10 000	~3 hours	0.000977	.976 milliseconds
1000 000	~11 days	0.0625	62.5 milliseconds

Floating point in the real world

- Using floating point to measure distances

FP value	Length	FP precision	Precision size
1	1 meter	1.19E-07	Virus
100	100 meter	7.63E-06	red blood cell
10 000	10 km	0.000977	toenail thickness
1000 000	.16x earth radius	0.0625	credit card width

Table source: Random ASCII

Floating point trouble

- Comparing floats for equality is a bad idea!

```
float f = 0.1;
while (f != 1.0) {
        f += 0.1;
}
```

```
f=0.2000000030
f=0.3000000119
f=0.4000000060
f=0.5000000000
f=0.6000000238
f=0.7000000477
f=0.8000000715
f=0.9000000954
f=1.0000001192
f=1.1000001431
f=1.2000001669
f=1.3000001907
f=1.4000002146
f=1.5000002384
f=1.6000002623
```

You are not alone in thinking FP is hard

- Many real world disasters are due to FP trickiness
  - Patriot Missile failed to intercept due to rounding error (1991)
  - Ariane 5 explosion due to overflow in converting from double to int (1996)

Floating point summary

- FP format is based on normalized exponential notation
- IEEE FP format
  - Normalized, denormalized, special values
- Floating points are tricky
  - Precision diminishes as magnitude grows
  - overflow, rounding error